Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/GB04/005459

International filing date: 30 December 2004 (30.12.2004)

Document type:	Certified copy of priority document	
Document details:	Country/Office:	GB
	Number:	0400148.3
	Filing date:	06 January 2004 (06.01.2004)

Date of receipt at the International Bureau: 25 February 2005 (25.02.2005)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)

PCT/GB2004/005459.

INVESTOR IN PEOPLE

The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.

Signed Stephen Hordley

Dated 11 February 2005

Patents Form 1/77

Patents Act 1977
(Rule 16)

THE PATENT OFFICE
CF
- 6 JAN 2004
RECEIVED BY FAX

The Patent Office

06JAN04 E863346-1 D10002
P01/7700 0.00-0400148.3 NONE

Request for grant of a patent

(See the notes on the back of this form. You can also get an explanatory leaflet from the Patent Office to help you fill in this form)

The Patent Office
Cardiff Road
Newport
South Wales
NP10 8QQ

1.	Your reference	jac.3186.uk.ja.d
2.	Patent application number *(The Patent Office will fill in this part)*	0400148.3 — 6 JAN 2004
3.	Full name, address and postcode of the or of each applicant *(underline all surnames)*	Calico Jack Ltd Argyll House Marketgait DUNDEE DD1 1QP
	Patents ADP number *(if you know it)*	8718546001
	If the applicant is a corporate body, give the country/state of its incorporation	UK
4.	Title of the invention	Dynamic modularity in flexible, persistent agents
5.	Name of your agent *(if you have one)* "Address for service" in the United Kingdom to which all correspondence should be sent *(including the postcode)*	Kennedys Patent Agency Limited Floor 5, Queens House 29 St Vincent Place Glasgow G1 2DT
	Patents ADP number *(if you know it)*	08058240002
6.	If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of these earlier applications and *(if you know it)* the or each application number	Country / Priority application number *(if you know it)* / Date of filing *(day / month / year)*
7.	If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application	Number of earlier application / Date of filing *(day / month / year)*
8.	Is a statement of inventorship and of right to grant of a patent required in support of this request? *(Answer 'Yes' if: a) any applicant named in part 3 is not an inventor, or b) there is an inventor who is not named as an applicant, or c) any named applicant is a corporate body. See note (d))*	Yes

0090664 06-Jan-04 04:53

Patents Form 1/77

9. Enter the number of sheets for any of the following items you are filing with this form. Do not count copies of the same document

Continuation sheets of this form	
Description	25
Claim *(s)*	CF
Abstract	
Drawing *(s)*	5 only

10. If you are also filing any of the following, state how many against each item.

Priority documents	
Translations of priority documents	
Statement of inventorship and right to grant of a patent *(Patents Form 7/77)*	
Request for preliminary examination and search *(Patents Form 9/77)*	
Request for substantive examination *(Patents Form 10/77)*	
Any other documents *(please specify)*	

11. I/We request the grant of a patent on the basis of this application.

Signature Kennedys KENNEDYS

Date 6 January 2004

12. Name and daytime telephone number of person to contact in the United Kingdom

Jim Adams Tel: 0141 226 6826

Warning

After an application for a patent has been filed, the Comptroller of the Patent Office will consider whether publication or communication of the invention should be prohibited or restricted under Section 22 of the Patents Act 1977. You will be informed if it is necessary to prohibit or restrict your invention in this way. Furthermore, if you live in the United Kingdom, Section 23 of the Patents Act 1977 stops you from applying for a patent abroad without first getting written permission from the Patent Office unless an application has been filed at least 6 weeks beforehand in the United Kingdom for a patent for the same invention and either no direction prohibiting publication or communication has been given, or any such direction has been revoked.

Notes

a) *If you need help to fill in this form or you have any questions, please contact the Patent Office on 08459 500505.*

b) *Write your answers in capital letters using black ink or you may type them.*

c) *If there is not enough space for all the relevant details on any part of this form, please continue on a separate sheet of paper and write "see continuation sheet" in the relevant part(s). Any continuation sheet should be attached to this form.*

d) *If you have answered 'Yes' Patents Form 7/77 will need to be filed.*

e) *Once you have filled in the form you must remember to sign and date it.*

f) *For details of the fee and ways to pay please contact the Patent Office.*

Dynamic Modularity in Flexible, Persistent Agents

The present invention relates to the dynamic deployment of functionality in software agents, in particular, a dynamic modular architecture for the agent that supports deployment of functionality that is not anticipated when the agent is instantiated.

Agents represent a means of representing a user in the electronic world, bringing together all the various functions that the user wants to perform in that electronic world, including:

- the transient, anonymous presence of an online search;
- persistent occasional presence of online shopping at a particular store;
- persistent passive presence of directed marking;
- persistent though temporary realtime presence in an online game;

and many more.

US Patent Application Publication Number US2002062334 to Hewlett Packard Company, CO, USA, entitled "Dynamic agents for dynamic service provision" teaches of dynamic

agents and a dynamic agent infrastructure that supports dynamic behaviour modification of agents. Dynamic agents carry out application specific actions, which can be loaded and modified on the fly. One key limitation of the approach presented is that it refers to the dynamic deployment of agent behaviour - but for the system to work as described, that dynamically deployed behaviour must be predefined (so that individual programs can know what to expect from others). The term 'dynamic' is thus misleading. The work could better be described as the dynamic deployment of statically defined behaviour. One problem presented by this aspect of US2002062334 is that software engineering in the large, and in particular, multi-party and multi-site development is made much harder by the need to conform to a rigid prespecified sets of functionality. It would be advantageous for new functionality to be definable dynamically, so that separate teams can worker in greater isolation, thereby simplifying problem decomposition.

US2002062334 describes dynamic functionality in an agent. It does not decouple functionality and therefore does not support functionality that is not at least defined at agent start-up. It discloses that the dynamic agent's modifiable portion includes its data, knowledge and action programs, which determines its application specific capabilities. Thus the generic, fixed portion of a dynamic agent, together with its application programs, acts as an autonomous problem solver. This means that generic capabilities (such as reasoning) are fixed at the point of startup of an agent. For agents that persist over a long period (months and years, rather than hours

or days) this means that it is impossible to introduce new, generic capabilities as they become available.

Finally, the US2002062334 does not explain how existing functionality might be replaced. Crucially, if demands are made of a carried action, and those demands pile up, and the carried action is then replaced, there is no way of coordinating the rescheduling of those demands for the new carried action. By having a specific component responsible for handover during update, it would become possible to queue these demands until the new functionality is in a position to handle them.

Agents represent individuals, and typically persist over long periods. It is thus a very real problem that adding new generic functionality and upgrading existing functionality in deployed systems should be prohibited. As a consequence, it would be advantageous to implement a method by which even the core, generic functionality can be replaced. A portion of the data of an agent may be retained, at the discretion of the update mechanism, to maintain coherence (so, for example, the beliefs of an agent may persist through an update of core functionality). This would allow upgrading of deployed agents through a straightforward, scalable mechanism.

US2002062334 discloses that when multiple actions are carried by the same dynamic agent, they can exchange information through the object store of that dynamic agent. A first problem with this is that for a carried action to be able to use data from another module, it must be able to anticipate the availability and internal structure of that data. That is, it must have knowledge

of the "interface" provided by that action. So for any two actions that might need to interact, they must (a) know that the other action is carried, (b) know enough about that action to know what data it can provide and in what format. In this way, any 'old' carried action will be unable to exploit the functionality developed in 'new' carried actions that was unanticipated at the time of the development of the 'old' carried action. This is a clear theoretical problem that translates into a large practical problem in situations where agent systems are deployed on a wide basis, particularly if they are persistent, and subject to rigorous quality of service requirements (such that they can't simply be 'switched off' for an upgrade). A second problem, in a similar vein, is that old actions cannot know of the methods (i.e. published procedures) of a newer carried action. It is presumably for this reason, coupled with the fact that it is much more difficult to design all possible method interfaces on a system-wide basis at the outset, than it is data interfaces, that means that the disclosed system does not support method calls between carried actions. This reduces the potential for synergistic interplay between carried actions, and can lead to requiring redundancy (reimplementing identical functionality in different carried actions). The problem is thus that one carried action cannot make a call upon the abilities of another. This violates key principles of modern programming practice aimed at code reuse and adaptation, by forcing modules to implement all the functionality that they will need, rather than utilising functionality they may know is already available within other modules in the agent. Agentative representation in mobile services represent a domain in which persistency is

Preferably said method means performs said function responsive to said request.

Preferably said request from a requesting module comprises a label specifying a function and said method means in said dockable module corresponds to the specified function.

Preferably said intermodule communication means comprises a store of labels and associated modules.

Preferably said store of labels and associated modules is a table.

Optionally said method means performs the function of docking said dockable module with said agent.

Optionally said method means performs the function of registering a label with said intermodule communication means, the label specifying a function supported by said dockable module.

Optionally said method means performs the function of undocking said dockable module from an agent.

Optionally said method means performs the function of requesting the discarding of a label from the intermodule communication means, the label specifying a function supported by said dockable module.

Optionally said method means performs the function of updating said dockable module within said agent.

According to a second aspect of the present invention, there is provided a method of deployment a module in an agent comprising the steps of:

- receiving a request for deployment of said module;
- maintaining registration information relating to a function supported by said module; and
- invoking said function.

Preferably said step of maintaining registration information comprises maintaining a store of labels and associated modules, each label specifying a function supported by a module.

Preferably said store of labels and associated modules is a table.

Preferably said step of maintaining registration information comprises the step of registering a label specifying a function supported by said module. This type of deployment is docking.

Alternatively said module is a docked module and said step of maintaining registration information comprises the step of unregistering a label specifying a function supported by said docked module. This type of deployment is undocking.

Alternatively said module is a replacement module and said step of maintaining registration information comprises the steps of:

- unregistering a label specifying a function supported by a docked module from said agent; and

- registering a label specifying a function supported by said replacement module with said agent

and said method further comprises the steps of:

- storing information related to the state of said docked module; and
- instantiating said replacement module using said stored information.

This type of deployment is updating.

Optionally, said step of maintaining registration information further comprises the step of queuing calls to the docked modules' registered labels, prior to the step of registering a label specifying a function supported by said replacement module.

According to a third aspect of the current invention said function supported by said module according to the second aspect is the method of deployment of said module according to the second aspect.

In order to provide a better understanding of the present invention, an embodiment will now be described by way of example only and with reference to the accompanying Figures, in which:

Figure 1 illustrates, in schematic form, an agent and a module in accordance with a preferred embodiment of the present invention;

Figure 2 illustrates, in schematic form, an overview of agentative representation in a multi-service environment;

Figure 3 illustrates, in schematic form, the process by which a label is resolved in accordance with a preferred embodiment of the present invention;

Figure 4 illustrates, in schematic form, module docking in accordance with the present invention;

Figure 5 illustrates, in schematic form, module updating in accordance with the present invention;

The inventions are an architecture and methods for dynamic deployment of modules in a software agent.

Although the embodiments of the invention described with reference to the drawings comprise computer apparatus and processes performed in computer apparatus, the invention also extends to computer programs, particularly computer programs on or in a carrier, adapted for putting the invention into practice. The program may be in the form of source code, object code, a code of intermediate source and object code such as in partially compiled form suitable for use in the implementation of the processes according to the invention. The carrier may be any entity or device capable of carrying the program.

For example, the carrier may comprise a storage medium, such as ROM, for example a CD ROM or a semiconductor ROM, or a magnetic recording medium, for example, floppy disc or hard disc. Further, the carrier may be a transmissible carrier such as an electrical or optical signal which may be conveyed via electrical or optical cable or by radio or other means.

When the program is embodied in a signal which may be conveyed directly by a cable or other device or means, the carrier may be constituted by such cable or other device or means.

Alternatively, the carrier may be an integrated circuit in which the program is embedded, the integrated circuit being adapted for performing, or for use in the performance of, the relevant processes.

With reference to Figure 1, the architecture 100 of an agent according to the present invention is best visualised as including a torus. On the inside of the torus 102, a special module, the core module 104, attaches itself. On the outside of the torus, any number of application specific modules 106, 108 may also become attached. The security and unity of the agent is also conceptually protected by a thin sphere 110 encompassing all the modules. The torus itself coordinates all communication between modules and between modules and core: this is the Inter Module Communication Layer (IMCL).

The modular architecture is implemented as a class that defines a module, from which specific classes, corresponding to modules with specific functionality, can be derived. A module 112 is shown with its components depicted. The module contains a number of components for implementing functions:

1. A docking method 114 that is called when the module is introduced to the agent and that describes the functionality provided by that module;

2. An undocking method 116 that is called when the module is required to detach itself from the agent and terminate;
3. An update method 118 that is called when the module is to be updated with a new version; and
4. A message handling method 120 for between-agent communication.

The docking method of a particular module provides the IMCL with a list of labels. The extensible set of these labels is consistent system-wide, functioning as the definition of the system's programming interface and is maintained centrally by the service provider. A particular module implements methods that match some (typically small) subset of all labels. On docking, a module informs the IMCL which labels it provides functionality for. As each module docks, the IMCL builds a table 122 of labels against which are listed the names of the methods that provide the functionality and the names of the modules implementing those methods.

In some cases, the labels implemented by a particular newly docking module may already be implemented by methods extant in the agent. So for example functionality f implemented by method x in module m1 may mean that when m1 docks in an agent, it needs to register the label f with the IMCL. But some other method, y, in some other module, m2, that is already docked, may also implement functionality f . Thus the IMCL would list functionality f has being implemented by m2.y.

In such situations, the IMCL adds the new label and associated method to its table, but also includes a

priority ranking, applied to both the new and old methods that implement the functionality. This priority is determined on the basis of some algorithm. Recency is a good example, whereby newer methods are always given a higher priority, but there are many such algorithms.

In addition, a module's docking method calls functions provided by the core's module management. These calls register information about the module with the core, including its name, provenance and labelled functionality. The processes of docking and module updating are described below with reference to Figures 7 and 8.

A user interacts with the electronic world for a host of reasons in a wide variety of domains: entertainment, e-commerce, professional, and so on. The present invention provides a means of bringing together all of these tasks and domains, and providing a single point of contact for the user, and allowing the sharing of user data between these different application domains. This contact is the user's agent, both in the computer-science sense (where agent oriented programming has particular restrictions, techniques and approaches, and places particular demands on software), and also in the intuitive sense of providing services of advocacy and representation. A user's agent is their permanent representative in the electronic world. Ideally, each user has exactly one agent, and a user's agent represents exactly one user (at the very least, such a relationship exists in a given context). The overall picture is as in Figure 2.

With reference to Figure 2, an overview of agentative representation in a multiservice environment is shown. The user 202 connects to their agent 206 at any time via any device (2G phones, multimedia mobile handsets, internet, etc.) in ways that are well known. The user agents 204 which represent users in the virtual world are shown. One user has a single agent 206 representing him or her in all their interactions in the virtual world. The service agents 208 provide specific services to any agents that request them, or that the service agents themselves decide to service. Information exchange between user and service agents can be initiated from either end. Some service agents 210 encapsulate existing legacy services (e.g., databases, Web Services and proprietary data handling systems). Broker agents 212 can mediate between a user and service agents. The user agents service agents and broker agents may be provided as a trusted service by a telecommunications operator.

An agent is a software entity with particular characteristics. We refer here to software processes that are:

(i) persistent (in that they continue to exist for an extended real time period, adapting to a single user over that time);

(ii) proactive (in that they include not only reactive behaviour, but also independently determined behaviour);

(iii) communicative (in that they communicate with other agents); and

(iv) autonomous (in that they typically cannot be directly modified by outside agencies, but must instead be altered through communciation).

The user can communicate with his agent across heterogeneous networks from a variety of devices, including mobile handsets and internet clients. In addition, however, the framework of the present invention supports the transparent filtering of information according to the device to which it is being sent. Thus the components within an agent that initiate communication with a user need not have any representation of the device type a user is employing. The content of the message is instead dynamically tailored to the user's device (e.g. summary text to an SMS-enabled mobile device, still pictures to a MMS-enabled mobile device, streaming video to broadband internet client platform, etc.).

The core is responsible for tailoring information to the device that is known to currently be available to the user. Thus, tailoring happens independently of the module calls, so that individual modules do not need to maintain device-specific information.

This filtering is achieved through a module-independent communication object that is filled in by individual modules when they need to communicate with the user. This object has subparts for different forms of media (text, picture, video, audio, etc.,). A module fills in as many of these subparts as it is able. The core then mediates the sending of that message to the user, by:

(i) identifying which device the user is currently employing (using a combination of historical usage patterns, presence information, and most recent-communication data);

User data (e.g., address; credit card details; age) and user preferences (e.g., policy on releasing credit card details; preference for aisle or window seat on planes; preferred DVD supplier) are stored in a local, private, secure database. Both user data and user preferences are extracted in three ways. First, through an explicit online interface that requests input on date of birth, or supports update to reflect change of address. Second, if the agent recognises information that it needs from the user, it can ask for it directly (e.g. asking a yes/no question by SMS). Third, as the user interacts with services manually, the agent can intercept information either explicitly or implicitly. If the user answers a particular question from a particular online service, the agent may either store that answer for future use, or ask the user explicitly if such storage is appropriate or useful. When acting autonomously, the agent provides information that external service requires (and no more), less anything that the user has placed a restriction on. Thus, for example, when interacting with an online newspaper, the newspaper provider may request user registration, but not demand it. In this case, the agent would provide no user information. Alternatively, when interacting with a book e-tailer, the e-tailer may require personal details including credit card data. If the user has instructed his or her agent not to give out credit card details without confirming it first, the agent would halt interaction with that site until user confirmation was sought and agreed.

These components could be represented by the steps:

1. Agent has goal of interacting with a service

2. Select required information from the user model (UM) (accesses the UM)
3. Check that the user model permits all this information to be freely given (accesses the UM)

If so,

4. Information given to the service

Otherwise

5. Process the restriction (either by terminating, or by asking the user, or by performing some other action)

The core also includes a subsystem responsible for passing messages to, and receiving messages from the user. The user may connect to his or her agent through a number of different channels: using a web browser on a PC, using a rich media mobile device (a Java phone, for example), using a high capacity mobile device (such as one that uses GPRS), or using an older, limited media device (say that can only handle voice and SMS traffic). The core implements labels that handle communication to and from such devices quite transparently: the calling module does not need to specify the different communication types at all.

The means by which one agent communicates with another is implemented in the core. Rather than supporting only agent-to-agent messages, the architecture is instead built around the idea that it is individual modules within agents that communicate with one another (this is "between agent module-module" or BAMM communication). Thus a module with expertise in buying in a particular e-commerce institution will communicate with a module in another agent that has expertise in selling in that same

labels with the IMCL and initialise 728 the BAMM handler before ending the process 730.

The undocking method of a particular module informs the IMCL and the core module management that the functionality provided by the module is terminating, and then kills the module's thread.

One of the advantages of decoupling of functionality in separate modules is in allowing modules to be added to an agent on-the fly. Such modules need not have functionality that is known before the agent is instantiated, as each module describes its own behaviour and functionality on docking. The same approach can be used to update existing modules with new versions.

Another, practical, advantage of the approach is that it removes compile time dependencies: a module developer can design, implement and test a module which makes calls to another module that they do not have, or do not have access to, or, indeed, that has not been developed at all. This simplifies many of the problems of software engineering in the large, and of multi-site collaborative development work.

The core is responsible for the handling of new modules and module updates. It arranges for outgoing requests to be sent for new modules (or for updated versions of existing modules), handles incoming requests for the agent to take on new modules, and handles the process by which a module is added. So, for example, if the user decides to sign up for a new, free service, the core may send a message to the coordinating agent for that

service, requesting the module. After some brief negotiation, the coordinating agent may send back a request to install the new module. The core module management accepts and downloads the software. After updating the internal module database, it starts the new module, which then docks with the IMCL, as described above.

The process of updating a module and the handover between the older and newer versions in illustrated in Figure 5. With reference to Figure 5, a flowchart 800 of the module updating process is shown. First, either a user specifies 802 a module, the core receives a message 804 specifying a module to update, or the core determines 806 that a module should be updated. The core reasons and decides 808 whether or not to update the module. If an update is determined to be completed, the core calls 810 the IMCL with the *getmodule* label and URI specifying the location of the Java Archive file. If not, the updating process ends 836. The IMCL then resolves 812 *updatemodule* label to the ModuleManagement.ModuleUpdate method in the core and the IMCL calls 814 the core's ModuleManagement.ModuleGet method with the module name and the URI. ModuleManagement calls 816 the module's UpdateMe method and the UpdateMe method returns the module state information, which is then stored 818 by ModuleManagement. ModuleManagement downloads 820 the Java Archive file from the URI. The .jar manifest file 822 specifies the Module class and the Module class 824 includes the DockMe method. With this information, the ModuleManagement instantiates 826 the Module class and calls 828 the DockMe method with the stored state information. The DockMe method may then register 830

labels with the IMCL and initialise 832 the BAMM handler before the Module Management calls 834 the UnDockMe method on the old module and unloads the class then ends the process 836.

The core's module management also handles more complex situations, in which a new version of an existing module is installed. The handover between the modules requires careful timing to ensure service to the rest of the agent are not disrupted. When a new version of a module is to be docked, the core module management calls the old version's update method, and, at an appropriate moment (at a break between discrete tasks that the module is working on) the module returns a representation of its current state.

Then, as part of the new module's docking method, it accepts the state of the previous version's data, so that it can pick up from where the previous module left off.

During the handover between versions of a module, the IMCL queues calls to the modules' labels, awaiting the docking of the new version.

Some of the functionality described herein is collected together in the core. Rather than hardwire a distinction between core and other modules, the core is instead implemented in just the same way as any other module. This means that calls on core functionality can exploit the robustness of the label-based, IMCL-mediated method calling.

It also means that core itself can be updated and docked through just the same process as any other module (so the activity of the module management component in the old core is interrupted and then taken up by the activity of the module management component in the new core). This means that new core functionality can be deployed dynamically, without other modules needing to know ahead of time all that might be included.

Further modifications and improvements may be added without departing from the scope of the invention herein described.

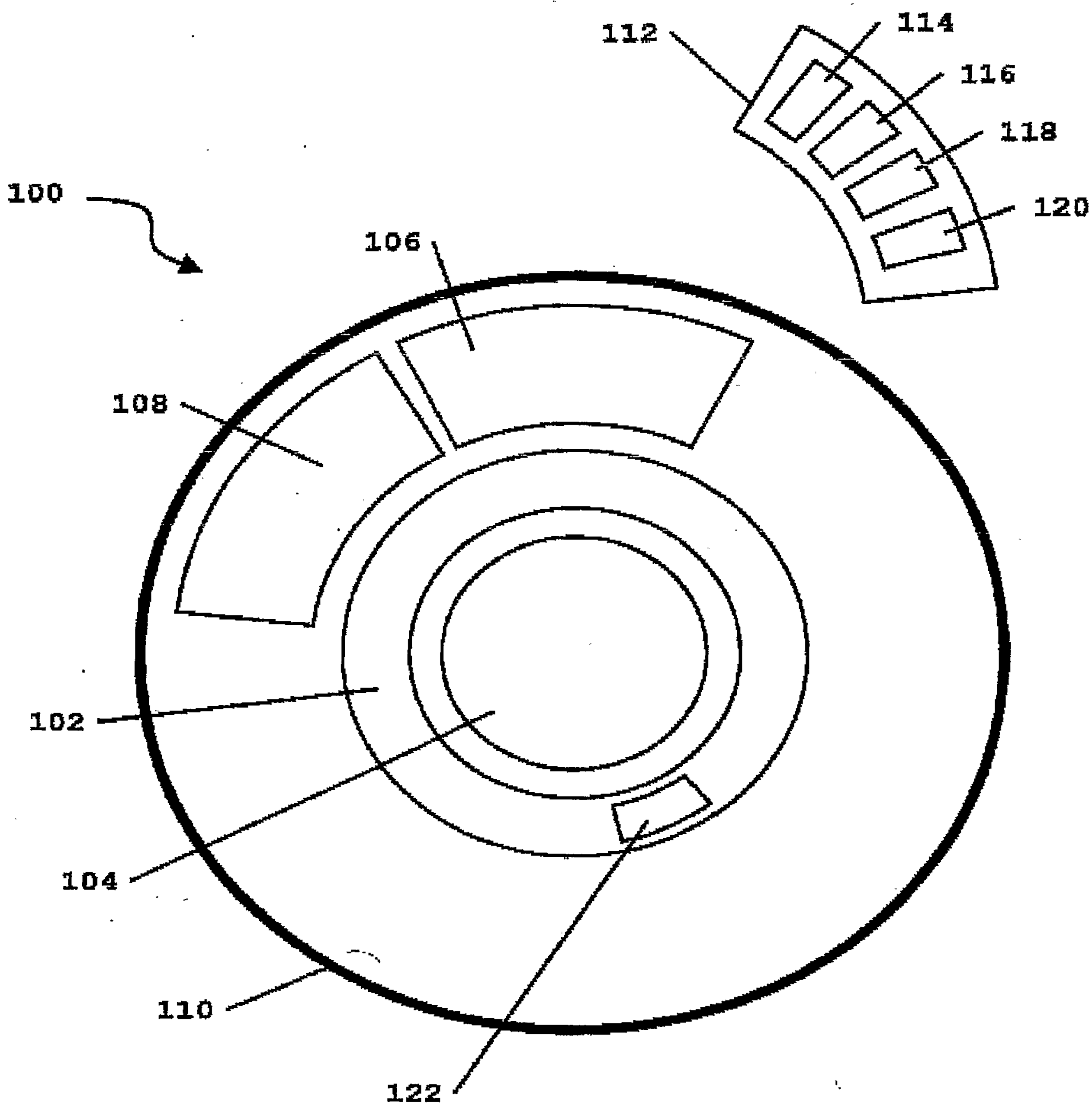

Fig. 1

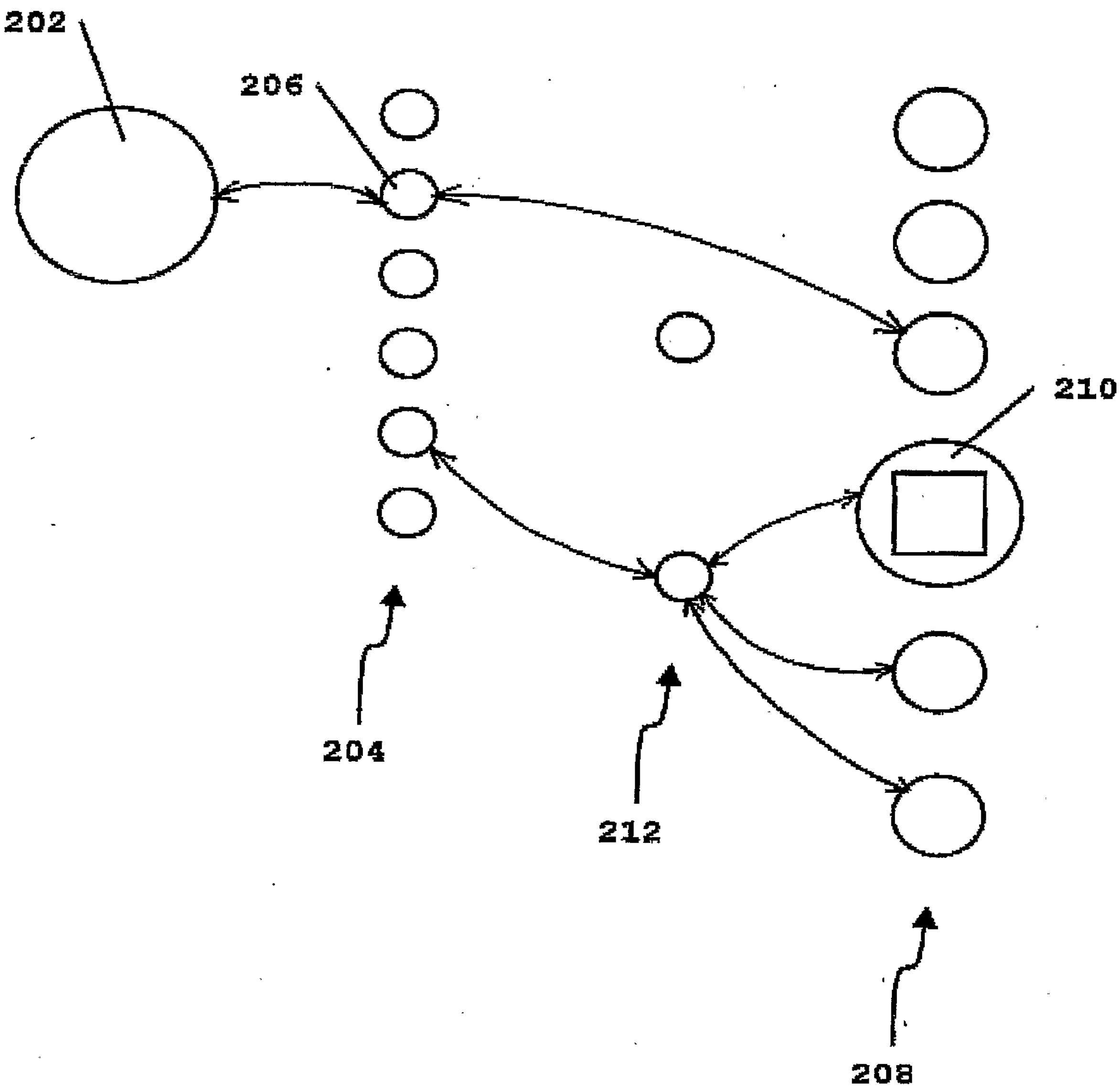

Fig. 2

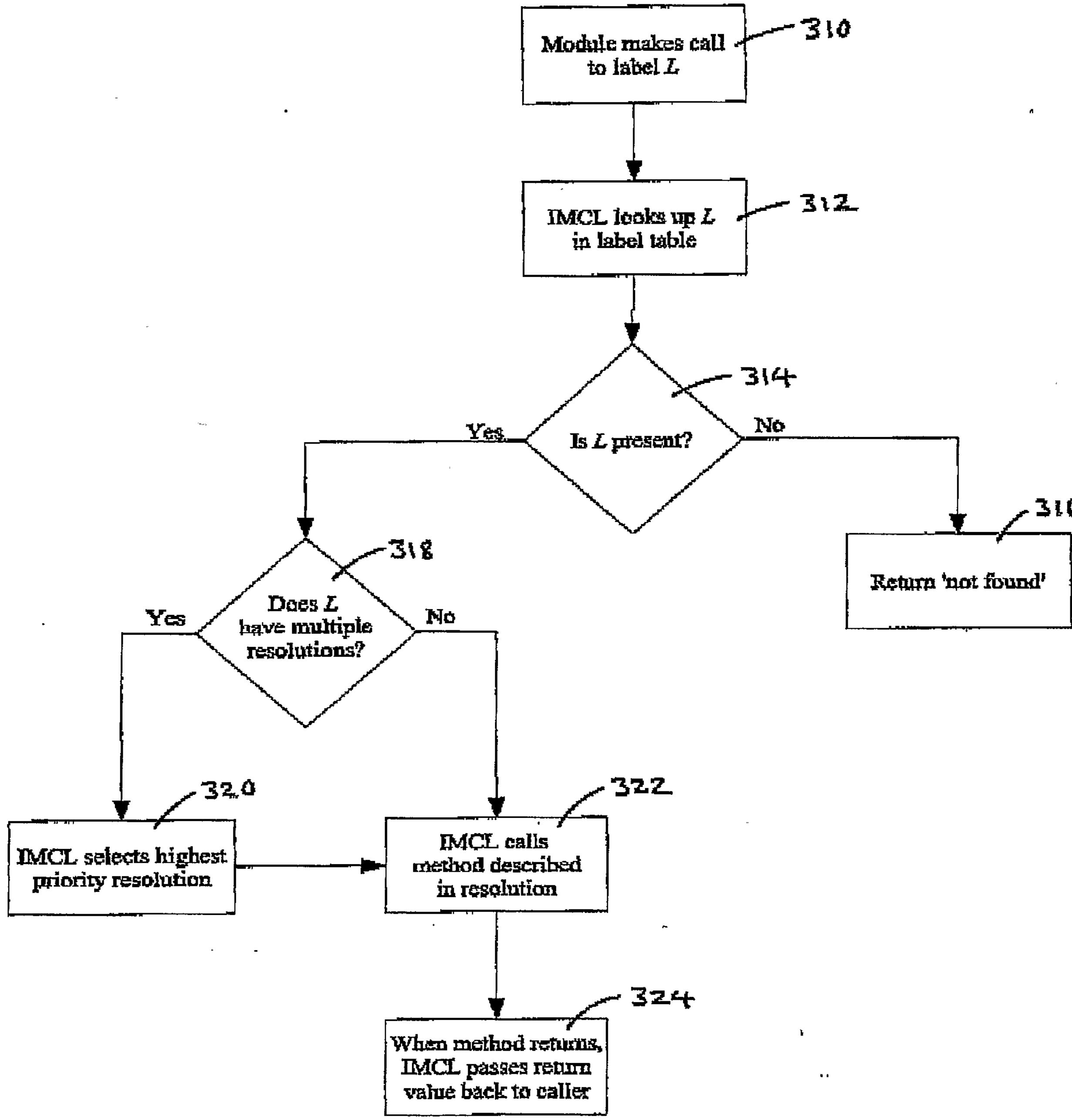
Module makes call to label L
310
IMCL looks up L in label table
312
Is L present?
314
Yes
No
Return 'not found'
316
Does L have multiple resolutions?
318
Yes
No
IMCL selects highest priority resolution
320
IMCL calls method described in resolution
322
When method returns, IMCL passes return value back to caller
324

Fig. 3

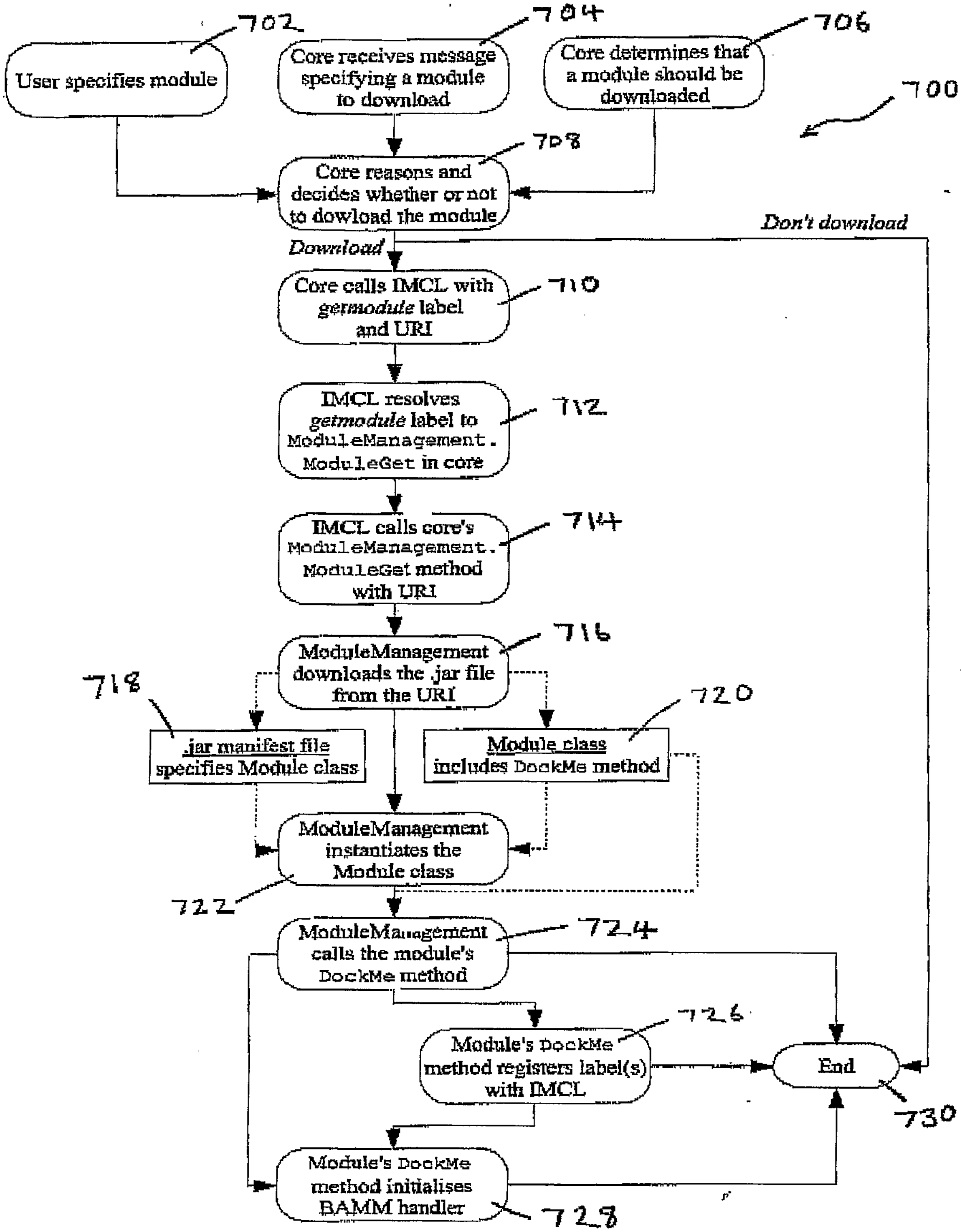
700
702
User specifies module
704
Core receives message specifying a module to download
706
Core determines that a module should be downloaded
708
Core reasons and decides whether or not to dowload the module
Don't download
Download
710
Core calls IMCL with getmodule label and URI
712
IMCL resolves getmodule label to ModuleManagement.ModuleGet in core
714
IMCL calls core's ModuleManagement.ModuleGet method with URI
716
ModuleManagement downloads the .jar file from the URI
718
.jar manifest file specifies Module class
720
Module class includes DockMe method
722
ModuleManagement instantiates the Module class
724
ModuleManagement calls the module's DockMe method
726
Module's DockMe method registers label(s) with IMCL
End
730
Module's DockMe method initialises BAMM handler
728

Figure 4

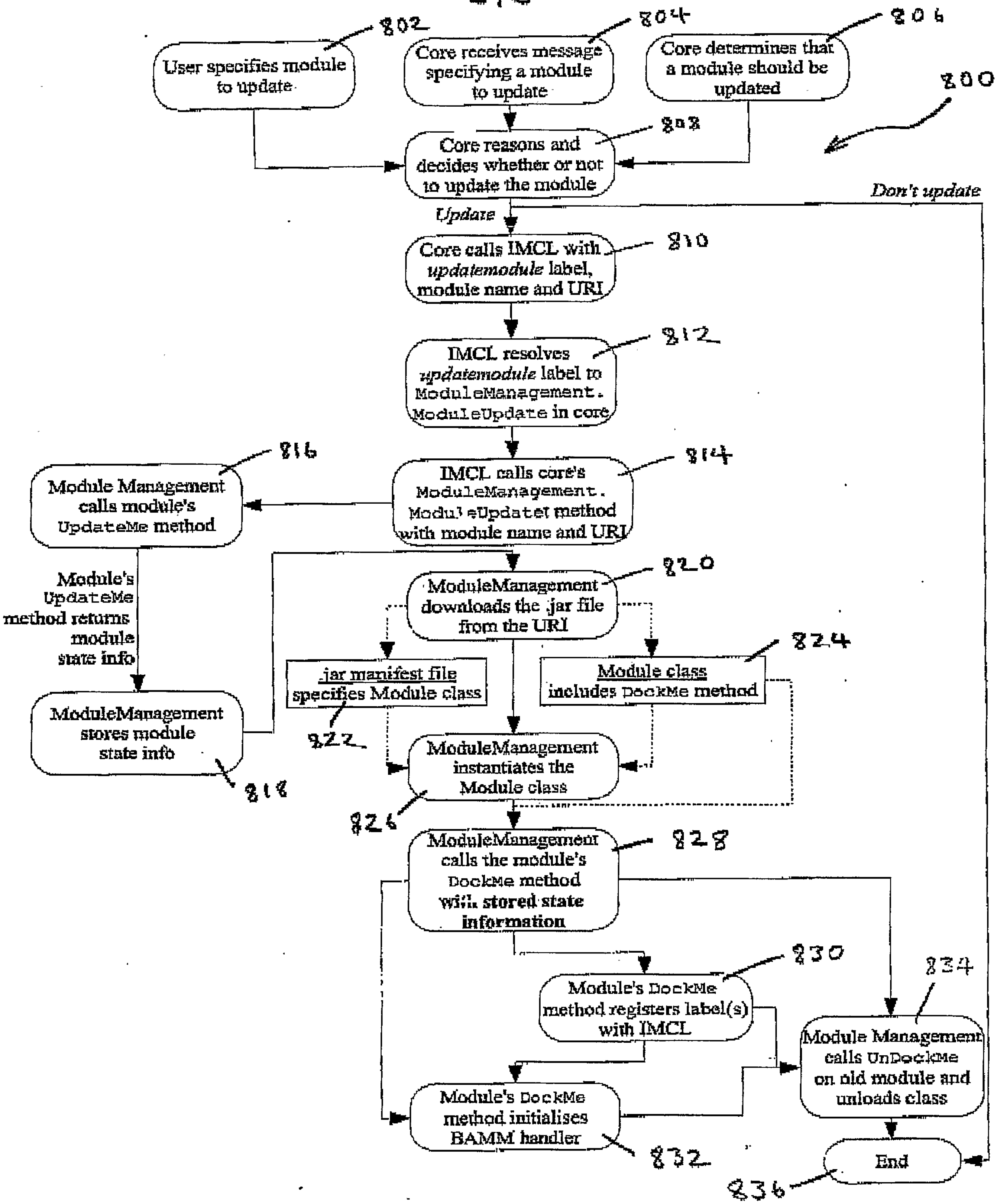
User specifies module to update
802
Core receives message specifying a module to update
804
Core determines that a module should be updated
806
800
Core reasons and decides whether or not to update the module
808
Don't update
Update
Core calls IMCL with *updatemodule* label, module name and URI
810
IMCL resolves *updatemodule* label to ModuleManagement. ModuleUpdate in core
812
Module Management calls module's UpdateMe method
816
IMCL calls core's ModuleManagement. ModuleUpdate method with module name and URI
814
Module's UpdateMe method returns module state info
ModuleManagement downloads the .jar file from the URI
820
.jar manifest file specifies Module class
822
Module class includes DockMe method
824
ModuleManagement stores module state info
818
ModuleManagement instantiates the Module class
826
ModuleManagement calls the module's DockMe method with stored state information
828
Module's DockMe method registers label(s) with IMCL
830
834
Module Management calls UnDockMe on old module and unloads class
Module's DockMe method initialises BAMM handler
832
End
836

Figure 5